Improving Tool Support for Software Reverse Engineering in a Security Context

Brendan Cleary[1], Christoph Treude[1], Fernando Figueira Filho[1], Margaret-Anne Storey[1], and Martin Salois[2]

[1] Dept. of Computer Science, University of Victoria
Victoria, BC, Canada
bcleary@uvic.ca

[2] Defence Research and Development Canada – Valcartier
Quebec, QC, Canada
martin.salois@drdc-rddc.gc.ca

Abstract. Illegal cyberspace activities are increasing rapidly and many software engineers are using reverse engineering methods to respond to attacks. The security-sensitive nature of these tasks, such as the understanding of malware or the decryption of encrypted content, brings unique challenges to reverse engineering: work has to be done offline, files can rarely be shared, time pressure is immense, and there is a lack of tool and process support for capturing and sharing the knowledge obtained while trying to understand assembly code. To help us gain an understanding of this reverse engineering work, we conducted an exploratory study at a government research and development organization to explore their work processes, tools, and artifacts [1]. We have been using these findings to improve visualization and collaboration features in assembly reverse engineering tools. In this talk, we will present a review of the findings from our study, and present prototypes we have developed to improve capturing and sharing knowledge while analyzing security concerns.

Keywords: malware, reverse engineering, empirical study.

1 Introduction

In his 1987 article [3], Cohen coined the term "computer virus" to describe self-reproducing programs designed to infect other computer programs. At that time, computer viruses were created for experimentation purposes or merely for fun, therefore causing little damage to real world systems [6].

Today's landscape shows us a different scenario. Computers are widely used in criminal activities such as bank fraud, identity theft, and corporate theft. According to a recent Symantec report [10], 2011 saw more than 187 million identities exposed in data breaches caused by hacking, and 93% more vulnerabilities related to mobile platforms—up to 315 in 2011 from 163 in 2010.

Illegal activities in cyberspace affect national security and threaten citizen's rights and privacy, thus having significant political, economic, and social implications [1]. Organized cyber groups typically communicate using cryptographic protocols and store information using encrypted files or systems. As a countermeasure against cybercrime, government institutions and business organizations have been using reverse engineering methods to analyze malicious code and break into password protected file systems.

This paper summarizes the findings of a first-of-its-kind field study we conducted with security engineers working in a government research and development organization [11] and looks at how we are incorporating these findings into tools designed to assist security engineers performing exploitability analysis [2].

Our study shows that security engineers have a unique work environment and experience significant challenges with urgency, documentation, and a limited ability to share information. Overall, security engineers have special needs in terms of time sensitivity, coordination, communication, and documentation.

2 Field Study Overflow

Our field study was conducted as an exploratory qualitative study. We conducted seven semi-structured interviews with engineers at a government research and development organization tasked with understanding targeted malware. For the remainder of this paper, we use P1 to P7 to refer to the participants of our study. A full description of the methodology can be found in [11]

To gain a comprehensive understanding of software reverse engineering in a government security context, our research questions focus on processes, tools, and artifacts:

1. What processes are part of reverse engineering in a security context?
2. What tools are being used?
3. What artifacts are being created and shared?

3 Summary of Findings

In this section, we present a summary of our findings from [11], subdivided for each research question posed in Section 2.

3.1 Processes

Based on the interview data, we identified five processes that are part of reverse engineering in a security context.

Analyzing. Analyzing assembly code is at the heart of most reverse engineering projects. Typical projects include the detection of malware, such as trojan horses, or the decryption of encrypted file systems. Assembly code is more difficult to understand than source code written in high-level programming languages because the code is less structured, often lacks meaningful symbols or data definitions, and allows for tricks that can mislead reverse engineers in their analysis efforts. Following the flow of data is challenging: *"Understanding the data flow is a big part of understanding a program."*$_{P4}$

Documenting. Documenting reverse engineering has several purposes. Some documentation is done to provide cognitive support for the reverse engineers at the time of the analysis, some documentation is meant to capture the reverse engineers' own understanding of the code, and other documentation is meant to be shared either with team members or outside stakeholders. While it is already difficult to document source code written in high-level programming languages, it is even more difficult when dealing with assembly code. During the exploration of the assembly code, most reverse engineers document just enough information to be able to resume a task and do not document the paths that were explored without success.

Transferring Knowledge. Transferring knowledge is a challenge in reverse engineering. Documentation alone is often not enough to understand the work that has been completed by somebody else: *"[I would] look at a version with comments, but I'd still need to jump through to understand."*$_{P7}$ In the current setting, information is usually passed on verbally or via email and chat. These mechanisms do not scale beyond groups of about five reverse engineers. To solve some of these issues, the idea of a workflow would be useful: *"Right now it's being done like a craft, and we'd like to have some kind of assembly line"*$_{P4}$. However, workflows are not consistent for all cases, and most workflow support tools are too constraining. In addition, documentation conventions and information sharing standards could improve the reverse engineering process: *"Respecting conventions [would make it] easier to pass from one project to another."*$_{P2}$

Articulating Work. Articulating work consists of all the items needed to coordinate a particular task, including scheduling sub-tasks, recovering from errors, and assembling resources [4]. In reverse engineering, where tangible results are only produced when a path of exploration is successful, constantly re-doing work is a problem. Work was usually divided based on different pieces of hardware, different vulnerabilities, different functions, or different files. Relating information from the analysis of different pieces of the problem was very difficult.

Reporting. When external stakeholders are involved, the final step in a project is reporting the results of the reverse engineering activities. In some cases, reporting includes a great deal of articulation work, especially when artifacts can be co-opted as reports: *"Instead of writing a report we shared a Word document."*$_{P6}$

3.2 Tools

Tools used by the participants in our study can be classified as disassemblers, office productivity and visualization tools, and communication and coordination tools.

Disassemblers. Most of the reverse engineering work is performed using IDA Pro[1]. IDA Pro is a commercial product that performs automatic code analysis and offers interactive functionality to support the understanding of disassembly. Reverse engineers typically start with an automatically generated disassembly listing, then rename and annotate sections in the listing until they understand the code. Debuggers are rarely used for malware in the early stages of analysis since portions of the code required for execution are often missing or because of the need to first remove anti-debugging tricks used by the malware. As one of our interviewees described it, the main analysis tool used by reverse engineers in the security context is *"brain power"*$_{P6}$.

Office Productivity and Visualization Tools. Most of the documentation is written using Microsoft Word, Excel, or OneNote. UML sequence diagrams are usually drawn to represent control flow understanding. However, the reverse engineers had *"trouble finding good tools that draw graphs and make it easy to navigate and export graphs"*$_{P1}$. Paper was also used, primarily for workflow support, small graphs, and articulation work.

Communication and Coordination Tools. For communication, only basic tools, such as e-mail and chat, were used. Our interviewees work in a co-located setting that allows face-to-face communication, but data sharing is complicated by the nature of the classified work. Interviewees coordinated work using tools such as wikis, bug trackers, and shared documents.

3.3 Artifacts

Artifacts created during the reverse engineering process in our setting consist of annotations, artifacts created for cognitive support, and reports.

Annotations. IDA Pro supports two notions of annotations: repeatable and non-repeatable. A repeatable annotation will appear attached to the current item as well as other items referencing it. Non-repeatable annotations only appear attached to the current item[2]. In addition, pre-comments and post-comments can be attached to lines and functions. All annotations also show up in the IDA Pro dependency graph.

The reverse engineers used annotations for several reasons: to keep track of variables, to rename functions, to document jumps, and to record where a particular piece of code was reading from or writing to. However, one of the challenges is that annotations are always incomplete: *"When you document stuff you tend to skip stuff that's obvious at the time."*$_{P6}$

Cognitive Support Artifacts. Depending on the use case, different documents are created by the reverse engineers to aid their cognition. These include:

[1] http://www.hex-rays.com/idapro
[2] http://www.hex-rays.com/idapro/idadoc/480.shtml

memory maps, Excel or Word tables showing register usage and boot processes, data flow diagrams, sequence diagrams, and scripts. A common scenario is when an engineer needs to keep track of different paths that are being explored in order to understand a particular piece of code. One of our interviewees used Microsoft OneNote to do that: *"I also used OneNote in other projects to keep track of paths that way. The last line in the OneNote document was the last path [that I had] explored."*$_{P6}$

Reports. Companies focused on malware, such as Symantec, frequently create reports that provide an overview of how a particular piece of malware works. Such reports rarely include enough detail to understand the inner workings of the malicious program, mostly because security companies do not want to reveal their insights to malware writers. In contrast, reports produced in our study setting had more technical content, and often included assembly code for functions as well as detailed descriptions of all input and output parameters.

4 Discussion of Challenges

Each work process described in the last section involved a different set of tools. These tools, in turn, were used to produce artifacts in distinct, non-interoperable formats. Therefore, moving from one process to another required a lot of manual work. By moving from the analysis to the documentation, engineers produce artifacts that would help them resume their own tasks, as well as transfer their knowledge to other team members. For example, reverse engineers have tried using wiki-based systems for sharing mixed content (e.g., details on how particular hardware works, including pieces of code). However, wikis have shortcomings when navigating code and related artifacts: *"Wikis are very document like, not ideal for documenting code – some kind of graph tool would have been better."*$_{P1}$. Overall, even when knowledge sharing was encouraged, reverse engineers faced a lack of proper tools to pass information along to others: *"There's also stuff that we don't know how to document."*$_{P1}$. Navigation is particularly a challenge when dealing with different documents such as the cognitive support artifacts mentioned above. A map of all documents and their connections usually only exists in the reverse engineer's head.

To articulate their work and break problems into pieces, engineers often followed a divide-and-conquer strategy: *"We go after different pieces. The problem is how to share information then... different people have different processes."*$_{P2}$. This poses an interesting phenomenon: there is no general process in the work of security reverse engineers. The following factors would influence this phenomenon:

Task Complexity. Tasks, such as blocking malware and breaking into secure devices, often include unsolved problems, thus requiring the use of different approaches, tools, and skills.

Security. The security context further obstructs the reverse engineers' work. Classified information cannot be easily shared, and for classified tasks, the reverse engineers are only allowed to work on classified, often un-networked, equipment. Often, information cannot be transported since it could belong to different projects, security classifications, or machines. Even for unclassified contexts, such as malware, the nature of the code prohibits easy sharing to prevent further infection. This also means that a lot of the work has to be completed offline, and access to web resources is very limited. Most of the reverse engineers in our study worked by themselves, often for security reasons: *"I'm the only one allowed to look at it [...] You don't want others to be infected [with malware]"*$_{P2}$.

Time Constraints. The amount of time pressure depends on the scenario. Some projects have the goal of understanding everything about a particular piece of software and are usually completed without time pressure. In other scenarios, only a couple of weeks are allocated for a particular project in order to provide a fast response to a potentially harmful threat. In the latter case, the reverse engineers have to prioritize what they are working on. In the example of malware: *"[We have] four goals when dealing with malware: detect, block, remove, [and] understand everything. Usually [the process] stops after the third step."*$_{P7}$ The amount of documentation produced depends on the extent of the time pressure. Long-term projects without time pressure yield more documentation, whereas for short-term projects, there is often not enough time to document thoroughly: *"If you put too much documentation, you won't have enough time to finish."*$_{P2}$

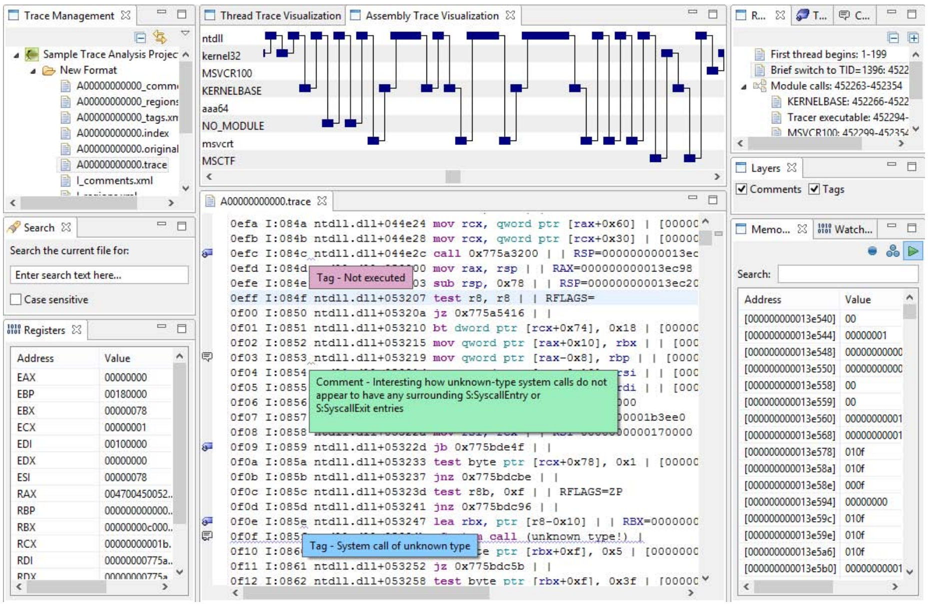
Tool Constraints. A graph is often the best way to capture a certain aspect of a reverse engineering problem, but it is difficult to deal with different types of diagrams. One of our interviewees told us that he sometimes spends up to 100 hours creating a single diagram. Also, the graphs produced are usually not linked to the disassembly, thus losing traceability. There is a shortage of tools that span different aspects of reverse engineering, such as hardware specifications and assembly code. The reverse engineering is also limited by memory since tools rarely scale beyond executables larger than a few megabytes.

5 Application of Findings

To demonstrate how these insights into the work practices of security engineers can be incorporated into tool design we, present Atlantis, a tool designed to assist software security engineers performing program exploitability analysis [2].

Exploitability analysis is the process of determining if a given program may be susceptible to exploitation. One way of determining if a program may have a hidden vulnerability is to: attempt to make the program crash (through a process called fuzzing [9]), trace the program (either by instrumention or an external tracer), and then analyze the resulting execution traces. While this process can be somewhat automated, assessing the actual exploitability of a crash and performing root cause analysis requires a great deal of human reasoning and manual analysis of the very large trace files generated.

Atlantis is an integrated assembly trace analysis environment designed to assist security engineers perform and manage this analysis. Atlantis was developed in collaboration with software security engineers to meet their requirements, and its design was informed by the work processes summarized in 3.1. Here we reuse these processes to structure our discussion of the features Atlantis offers.



```
0efa I:084a ntdll.dll+044e24 mov rcx, qword ptr [rax+0x60] | [00000C
0efb I:084b ntdll.dll+044e28 mov rcx, qword ptr [rcx+0x30] | [00000C
0efc I:084c ntdll.dll+044e2c call 0x775a3200 | | RSP=00000000013ec
0efd I:084d                00 mov rax, rsp | | RAX=00000000013ec98
0efe I:084e                03 sub rsp, 0x78 | | RSP=00000000013ec20
0eff I:084f ntdll.dll+053207 test r8, r8 | | RFLAGS=
0f00 I:0850 ntdll.dll+05320a jz 0x775a5416 | |
0f01 I:0851 ntdll.dll+053210 bt dword ptr [rcx+0x74], 0x18 | [00000C
0f02 I:0852 ntdll.dll+053215 mov qword ptr [rax+0x10], rbx | | [000
0f03 I:0853 ntdll.dll+053219 mov qword ptr [rax-0x8], rbp | | [0000
0f04 I:0854                                              rsi | | [000
0f05 I:0855                                              rdi | | [000
0f06 I:0856                                              000
0f07 I:0857                                              00001b3ee0
0f08 I:0858                                              0000170000
0f09 I:0859 ntdll.dll+05322d jb 0x775bde4f | |
0f0a I:085a ntdll.dll+053233 test byte ptr [rcx+0x78], 0x1 | [00000C
0f0b I:085b ntdll.dll+053237 jnz 0x775bdcbe | |
0f0c I:085c ntdll.dll+05323d test r8b, 0xf | | RFLAGS=ZP
0f0d I:085d ntdll.dll+053241 jnz 0x775bdc96 | |
0f0e I:085e ntdll.dll+053247 lea rbx, ptr [r8-0x10] | | RBX=0000000
0f0f I:085                                 call (unknown type!) |
0f10 I:086                         e ptr [rbx+0xf], 0x5 | [0000000
0f11 I:0861 ntdll.dll+053252 jz 0x775bdc5b | |
0f12 I:0862 ntdll.dll+053258 test byte ptr [rbx+0xf], 0x3f | [00000
```

Tag - Not executed

Comment - Interesting how unknown-type system calls do not appear to have any surrounding S:SyscallEntry or S:SyscallExit entries

Tag - System call of unknown type

First thread begins: 1-199
Brief switch to TID=1396: 4522
Module calls: 452263-452354
KERNELBASE: 452266-4522
Tracer executable: 452294-
MSVCR100: 452299-452354

Address	Value
EAX	00000000
EBP	00180000
EBX	00000078
ECX	00000001
EDI	00100000
EDX	00000000
ESI	00000078
RAX	004700450052..
RBP	000000000000..
RBX	00000000c000..
RCX	00000000001b.
RDI	00000000775a..
RDX	00000000775a

Address	Value
[000000000013e540]	00
[000000000013e544]	00000001
[000000000013e548]	0000000000
[000000000013e550]	0000000000
[000000000013e558]	00
[000000000013e559]	00
[000000000013e560]	0000000001
[000000000013e568]	0000000001
[000000000013e578]	010f
[000000000013e58a]	010f
[000000000013e58e]	000f
[000000000013e594]	00000000
[000000000013e59c]	010f
[000000000013e59e]	010f
[000000000013e5a6]	010f
[000000000013e5b0]	0000000001

Fig. 1. Atlantis

5.1 Analyzing

With the advent of new tracing technologies (e.g. BitBlaze [7], Pin [5]) we believe analyzing trace files will become the primary task performed by engineers when conducting exploitability analysis of a program. Currently, engineers rely on off-the-shelf text editors and file comparison tools to analyze traces. Atlantis (Figure 1) improves on these tools by providing three customized and linked views: a dedicated Trace Text Viewer, a Trace Visualization View, and a Memory State View which work in concert to allow the engineer to perform their analysis.

1. The **Trace Text Viewer** is a simple text viewer and comparison tool but improves on off-the-shelf solutions with features like very large file support,

fast search, trace-specific syntax highlighting, and memory reference highlighting.
2. The **Trace Visualization View** provides engineers with a high-level representation of the trace. It was designed to help navigate very large traces and to provide a visual overview of the entire trace under study.
3. The **Memory State View** (using an innovative indexing approach) allows an engineer to reconstruct the entire memory state of the program under study at any point in the execution trace, in real time.

5.2 Documenting and Reporting

Documenting trace files (like documenting source code or disassembly) is a similarly crucial part of exploitability analysis. Engineers document traces both to support their analysis activities and to capture and share their findings with other engineers. Atlantis supports this process by providing rich annotation features, allowing engineers to attach comments and tags to locations within the trace.

The Comments View and Tags View provide a way for users to quickly record hypotheses as they traverse the trace. Building on previous work on tagging in software development [8], tags allow a user to annotate a particular line and column (or entire sections) in the trace. There can be multiple occurrences of a tag and using the Tags View, the user can navigate between all occurrences of a tag. Tags can also be grouped into different sets and labeled. Comments function in a similar way but are unique and allow a user to express more complex ideas about a particular location or section of the trace.

Unlike traditional source code editors, where comments and tags are expressed in-line with the source, in Atlantis, comments and tags are displayed in a separate UI layer floating above the Trace Text Viewer and are stored in separate files. This allows the user to selectively display only particular groups of comments and tags. For example, a user analyzing a trace might have different comment groups for different features they are investigating in the trace. Comment and tag layering allows a user to quickly show or hide all comments from one or both of those features.

5.3 Transferring Knowledge

Exploitability analysis offers a lot of opportunity for engineers to collaborate when analyzing traces. Atlantis supports collaboration between engineers by storing annotations separate from traces, and by making it easier for them to share and manage trace annotations. This has multiple benefits.

1. Engineers don't have to share the trace files themselves, but rather just the annotation files. This can be a significant benefit due to the large size of the trace files.
2. As the annotation files are simple xml files, engineers can place them under version control, allowing them to version and share annotations with other engineers in an organized and traceable fashion.

3. If multiple engineers are analyzing a trace collaboratively, they can easily merge annotations from other engineers into their own annotations and then re-export their annotations to be shared with the group.

5.4 Articulating Work

When performing exploitability analysis, engineers will typically not be working with just a single trace, but rather a set of multiple types of traces. For example, along with 'failing' traces that result from program crashes (and which may demonstrate an exploit), engineers often want to analyze and compare 'passing' traces (traces which demonstrated correct operation of the program). The Atlantis Project Management View provides engineers a mechanism for organizing their exploitability analysis of a program into a project structure (including trace and annotation files) and to share that project structure with other engineers through version control. This allows engineers to treat the exploitability analysis of a program as a coherent, standardized entity in itself, rather than as a disparate collection of trace files and documentation.

6 Conclusion and Future Work

The work setting of reverse engineers tasked with security-related issues, such as the dissection of malware or the decryption of encrypted file systems, is unique. Web resources are often unavailable because work has to be performed offline, files can rarely be shared to avoid infecting co-workers with malware or because information is classified, time pressure is immense, and tool support is limited.

In this paper we presented an overview of an exploratory study we conducted [11] to gain an understanding of the work done by security reverse engineers and to understand their processes, tools, artifacts, challenges, and needs. We also reported on Atlantis, a tool that attempts to incorporate the findings of that study and is designed to assist software security engineers with identifying potentially exploitable programs based on analysis of their execution traces. Reverse engineering in a security context is a fast-changing environment. New tools and approaches have to be learned on the spot as hackers and organized cyber groups constantly create new security threats with implications for national security. Future work lies in addressing the challenges that we have identified with improved tools and processes, and in studying their usefulness in the unique work environment of security reverse engineers.

Acknowledgments. We wish to thank the participants in this study, and Cassandra Petrachenko for her feedback on this paper. This research is funded through NSERC grant DNDPJ 380607-09 and DRDC Valcartier.

References

1. Choo, K.K.: Organised crime groups in cyberspace: a typology. Trends in Organized Crime 11, 270–295 (2008)
2. Cleary, B., Painchaud, F., Chan, L., Storey, M.A., Salois, M.: Atlantis - assembly trace analysis environment. In: IEEE 19th Working Conference on Reverse Engineering, WCRE 2012 (2012)
3. Cohen, F.: Computer viruses: Theory and experiments. Computers & Security 6(1), 22–35 (1987)
4. Gerson, E.M., Star, S.L.: Analyzing due process in the workplace. ACM Transactions on Information Systems 4, 257–270 (1986)
5. Luk, C.K., Cohn, R., Muth, R., Patil, H., Klauser, A., Lowney, G., Wallace, S., Reddi, V.J., Hazelwood, K.: Pin: building customized program analysis tools with dynamic instrumentation. In: Proceedings of the 2005 ACM SIGPLAN Conference on Programming Language Design and Implementation, PLDI 2005, pp. 190–200 (2005)
6. Peterson, T.F.: A History of Hacks and Pranks at MIT. The MIT Press (2011)
7. Song, D., Brumley, D., Caballero, J., Jager, I., Kang, M.G., Liang, Z., Newsome, J., Poosankam, P., Saxena, P.: Bitblaze: A new approach to computer security via binary analysis. In: Proceedings of the 4th International Conference on Information Systems Security (2008)
8. Storey, M.A., Ryall, J., Singer, J., Myers, D., Cheng, L.-T., Muller, M.: How software developers use tagging to support reminding and refinding. IEEE Transactions on Software Engineering 43 (2009)
9. Sutton, M., Greene, A., Amin, P.: Fuzzing: Brute Force Vulnerability Discovery. Addison-Wesley (2007)
10. Symantec: Internet security threat report, vol. 17 (April 2012), `http://bit.ly/15nJXO7` (last access: January 3, 2012)
11. Treude, C., Figueira Filho, F., Storey, M.A., Salois, M.: An exploratory study of software reverse engineering in a security context. In: 18th Working Conference on Reverse Engineering (WCRE 2011), pp. 184–188 (2011)